ReactJS.

The basics

ReactJS.

- **A Javascript Single Page App (SPA) framework for building dynamic Web** User Interfaces**.**

- **Client-side framework.**
  - **More a library than a framework.**

Before ReactJS.

- **Traditional SPAs promoted the MVC pattern for** app design.
  - **Ex: AngularJS, EmberJS, BackboneJS etc.**
- **React is not MVC, just V (and maybe some C).**
  - **It challenged established best practice (MVC).**
- **MVC frameworks use** template **technology at the V layer; React based on** components

	Templates	React components
Separation of concerns	Technology (JS, HTML)	Responsibility "Display UI"
Semantic	New concepts and micro-languages	HTML and Javascript
Expressiveness	Underpowered	Full power of Javascript

ReactJS

- **Philosophy:** *Build components, not templates*
- **All about the UI (User Interface).**
  - **Not about business logic or the data model.**
- **Component: A unit comprised of HTML(ish), JS, and data.**
  - **UI description (HTML) and UI behavior (JS) are** tightly coupled **and** co-located**.**
    - **Other SPA frameworks decoupled these aspects.**
- **Benefits:**
  1. **Composability.**
  2. **Reusability.**
  3. **Performance.**

Creating the UI description

- React.createElement() **– create a DOM element.**
- ReactDOM.render() –**attach an element to the current DOM.**
- **Ref. 01-**UIDescription.html

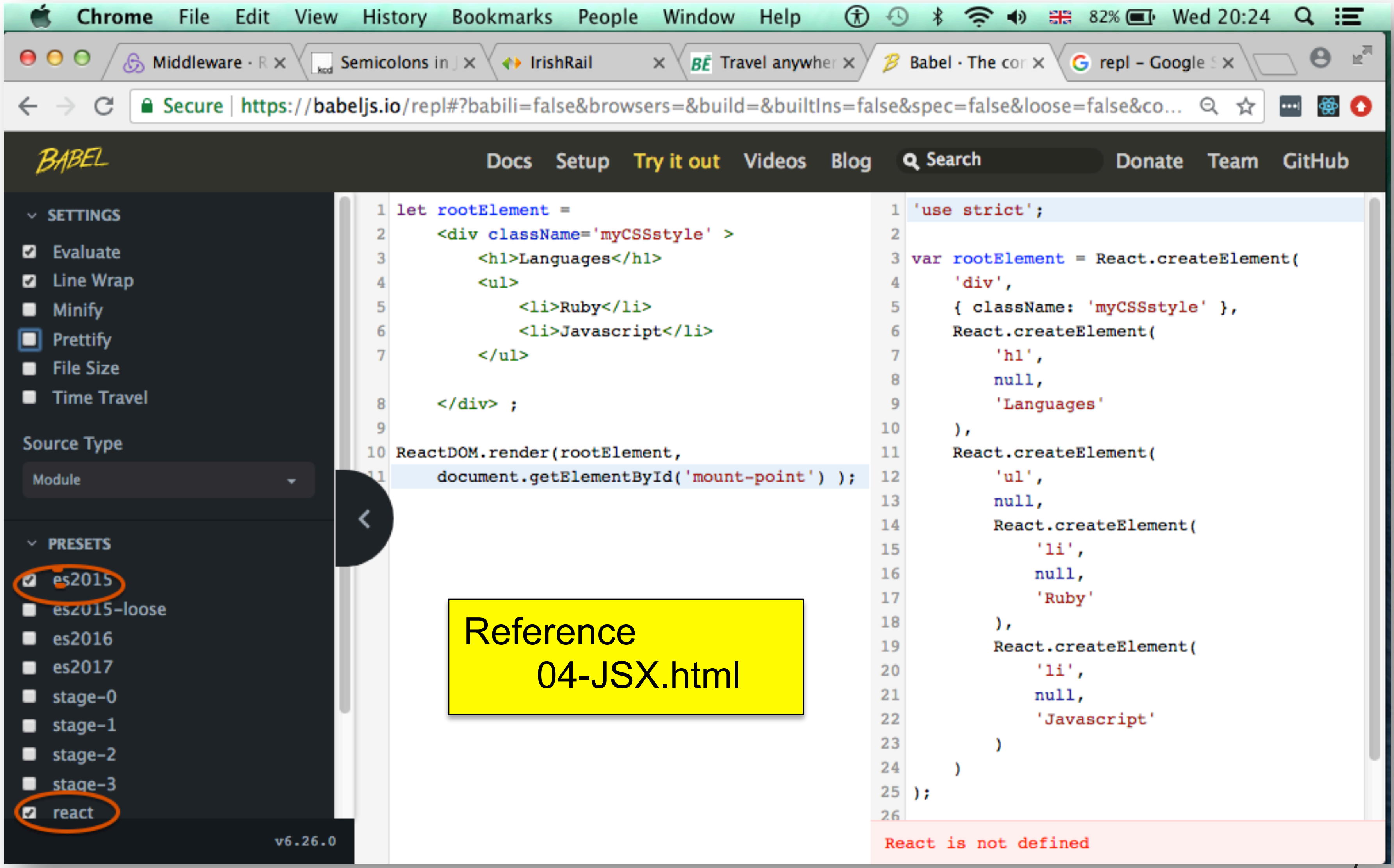
- createElement() **arguments**:
  - **type (h1, div, span etc);**
  - **properties (style, event handler etc);**
  - **children.**
- render() **arguments:**
  - **element to be displayed;**
  - **DOM node on which to** mount **the element**.
- **Ref**. 02-UIDescription.html

- **We never use** createElement**() directly – far too cumbersome.**

JSX.

- **JSX – JavaScript extension syntax**
- **Declarative syntax for coding UI descriptions.**
- **Retains the full power of Javascript.**
- **Allows tight coupling between UI logic and description.**
- **Must be transpiled (Babel) for browser execution compatibility.**
  - **Reference** 03-JSX-error.html

REPL (Read-Evaluate-Print-Loop) transpiler.



Chrome File Edit View History Bookmarks People Window Help
Wed 20:24
https://babeljs.io/repl#?babili=false&browsers=&build=&builtIns=false&spec=false&loose=false&co...
BABEL
Docs Setup Try it out Videos Blog Search Donate Team GitHub
SETTINGS
Source Type
PRESETS
v6.26.0

Settings: [✓ Evaluate] [✓ Line Wrap] Minify, Prettify, File Size, Time Travel

Source Type: Module

Presets: [✓ es2015] es2015-loose, es2016, es2017, stage-0, stage-1, stage-2, stage-3, [✓ react]

```
let rootElement =
    <div className='myCSSstyle' >
        <h1>Languages</h1>
        <ul>
            <li>Ruby</li>
            <li>Javascript</li>
        </ul>

    </div> ;

ReactDOM.render(rootElement,
    document.getElementById('mount-point') );
```

```
'use strict';

var rootElement = React.createElement(
    'div',
    { className: 'myCSSstyle' },
    React.createElement(
        'h1',
        null,
        'Languages'
    ),
    React.createElement(
        'ul',
        null,
        React.createElement(
            'li',
            null,
            'Ruby'
        ),
        React.createElement(
            'li',
            null,
            'Javascript'
        )
    )
);
```

```
React is not defined
```

Reference
04-JSX.html

JSX.

- **HTML-like markup.**
- **It’s actually XML code.**
- **Must be transformed (transpiled) into ES5.**
  - **The Babel tool suite.**
- **Some minor HTML tag attributes differences, e.g. className (class), htmlFor (for).**
- **Allows** declarative description **of the UI inlined in JavaScript.**
- **Combines the ease-of-use of templates with the power of JS.**

Transpiling JSX.

- **What?**
  - **The Babel platform**

- **How?**
  1. **Manually, via REPL or command line.**
     - **When experimenting only.**
  2. **By the web server (using special tooling, i.e.Webpack).**
     - **Suitable for app development mode.**
  3. **As part of the build process for an app.**
     - **When deploying app for production.**

React Components.

- **We develop COMPONENTS.**
  - **A** class **that extend** React.Component
  - The render() **method:**
    - **Mandatory.**
    - **Returns the component** UI description**.**

- **We reference components as HTML tags - <ComponentName>**
  - **E.g.** ReactDOM.render(<ComponentName/>, ...)

- **Reference** 05-simpleComponent.html

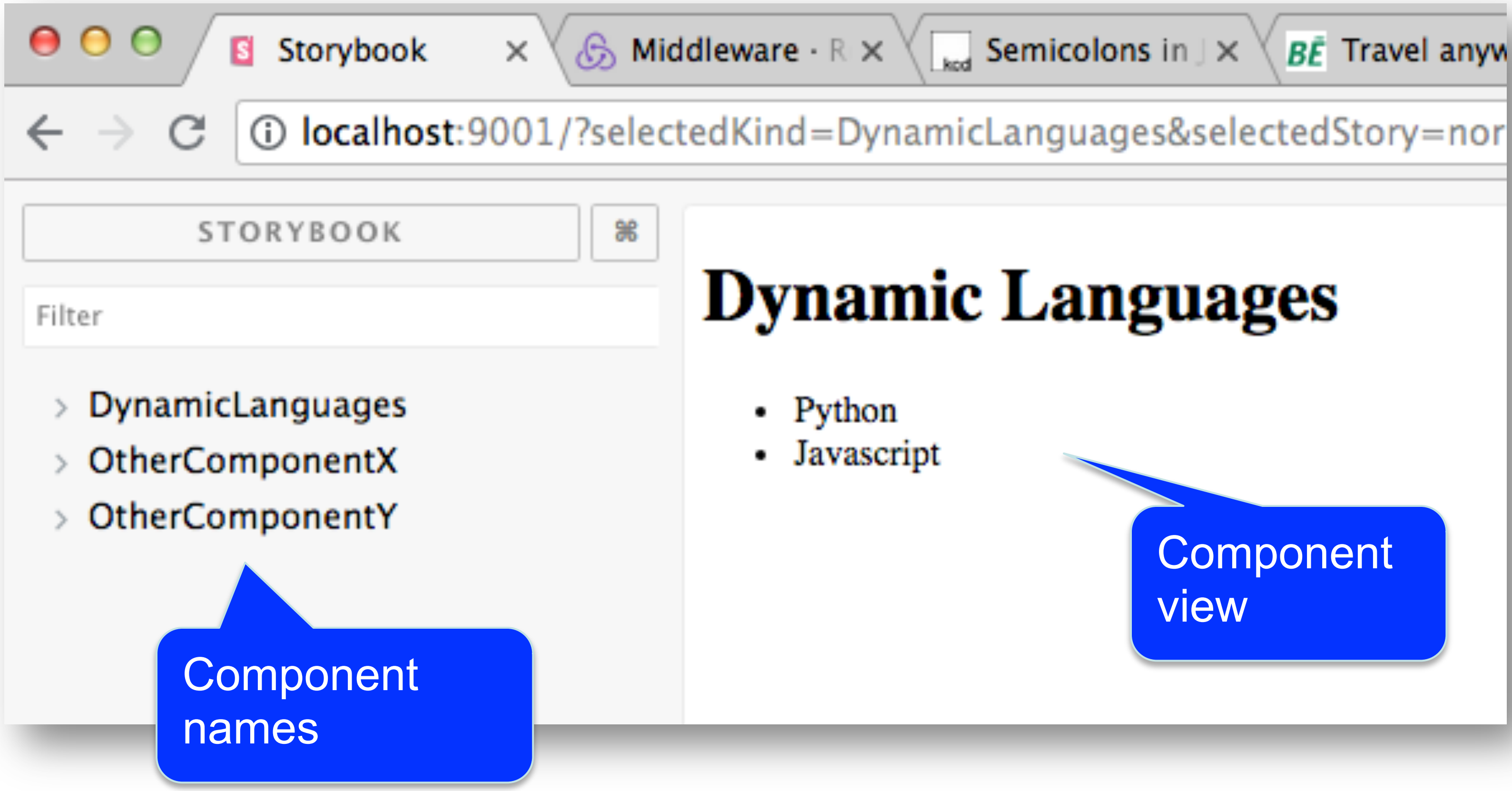
React Development tools.

- **The** create-react-app **tool. Features:**
  - **a) Scaffolding/Generator**
  - **b) Development web server – auto transpilation on file change + live reloading.**
  - **c) Builder: build production standard version of app, i.e. minification, bundling.**

- **The** Storybook **tool:**
  - **A** development environment **for UI components**
  - **Runs outside of your app - develop component in isolation.**
  - **Leads to more reusable, testable components**
  - **Quicker development – ignore application-specific dependencies.**

STORYBOOK

- **Tool interface.**


Storybook
Middleware · R
Semicolons in J
Travel anyw
localhost:9001/?selectedKind=DynamicLanguages&selectedStory=nor
STORYBOOK
Filter
DynamicLanguages
OtherComponentX
OtherComponentY

Dynamic Languages

- Python
- Javascript

STORYBOOK

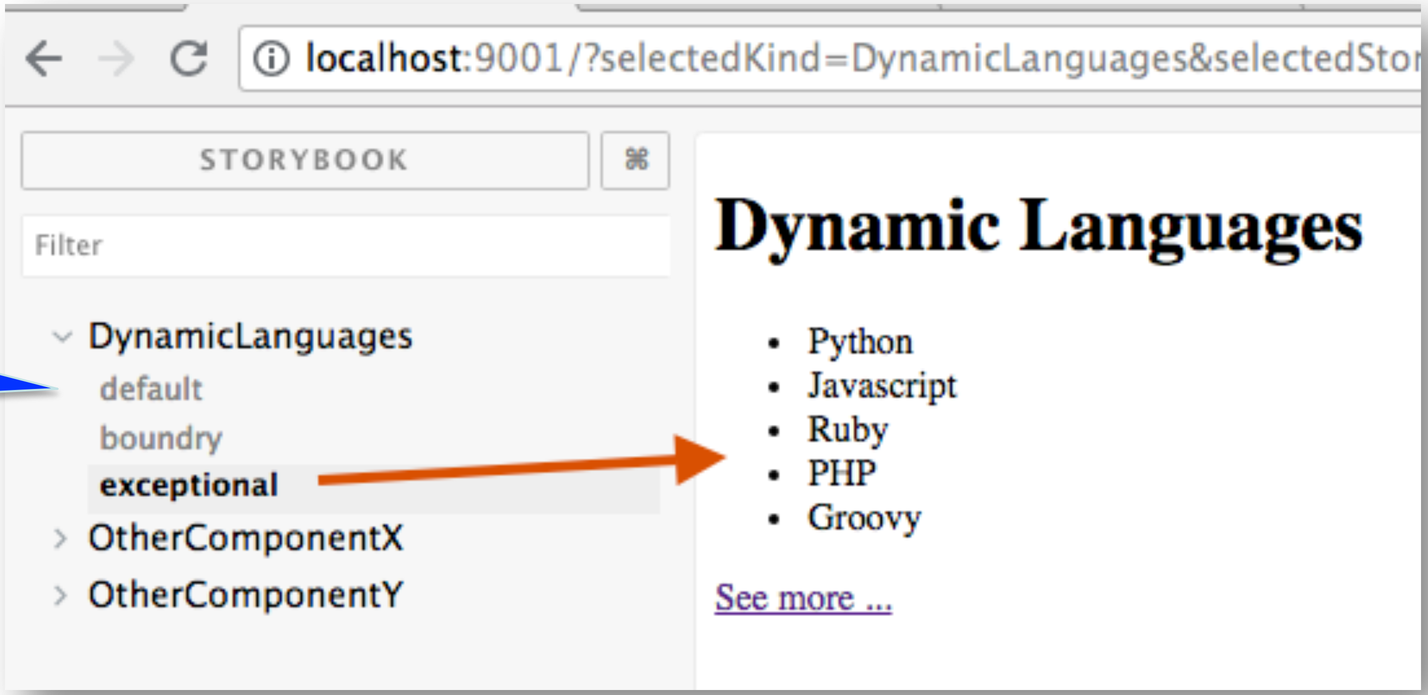
- **Component design considerations:**
  - **A component may have several STATES → The state influences how it should render,**

- EX.: DynamicLanguages **component:**
  - **Default – list of languages less than 5 → Renders full list**
  - **Boundary – empty list → Render 'No languauge' message**
  - **Exceptional – More than 5 languages → Render first 5 and a 'See More…' link to display next 5.**

- **Each state case termed a STORY**

STORYBOOK

- **A component may have many stories.**
- **The tool allows us** document them.



```
localhost:9001/?selectedKind=DynamicLanguages&selectedStor
```

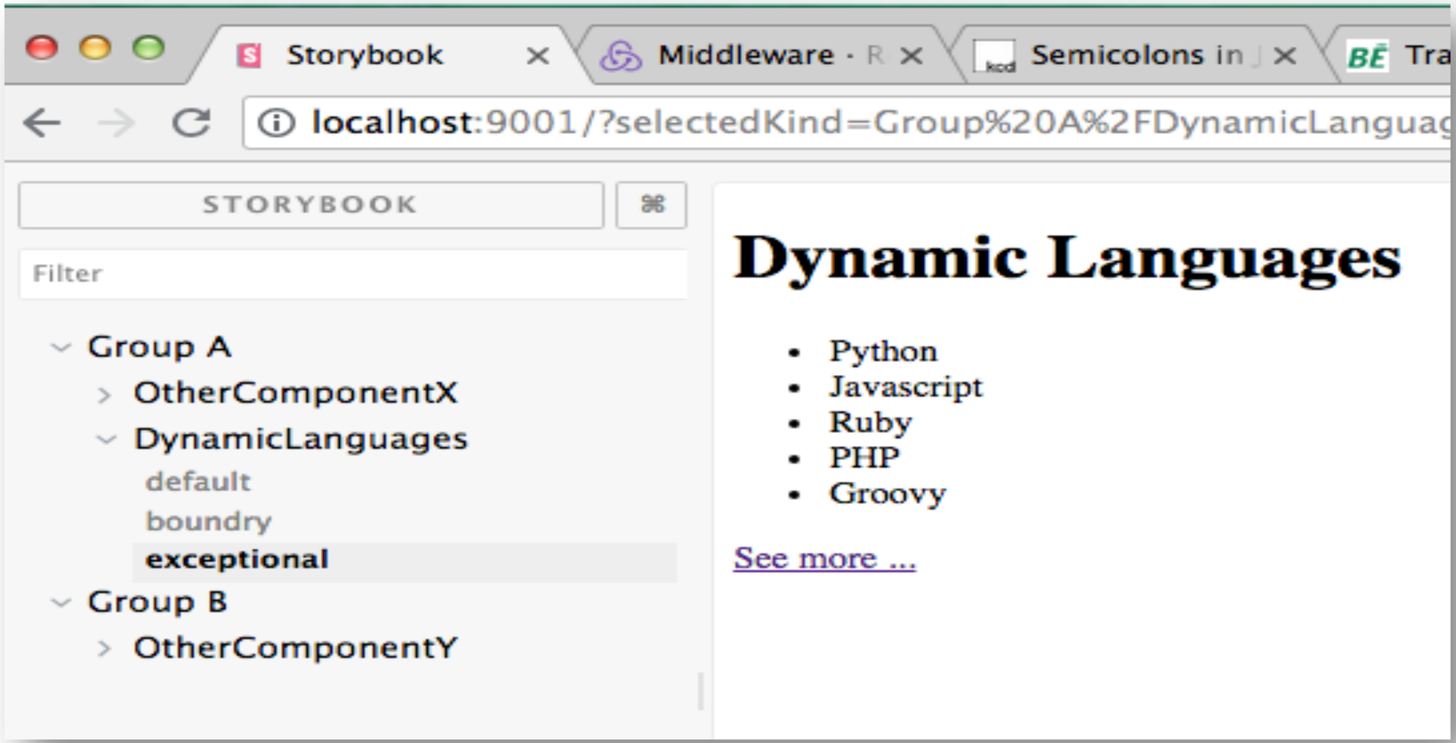
Dynamic Languages

- Python
- Javascript
- Ruby
- PHP
- Groovy

See more ...

STORYBOOK

- **For large component libraries,** grouping **helps others understanding the catalogue**.



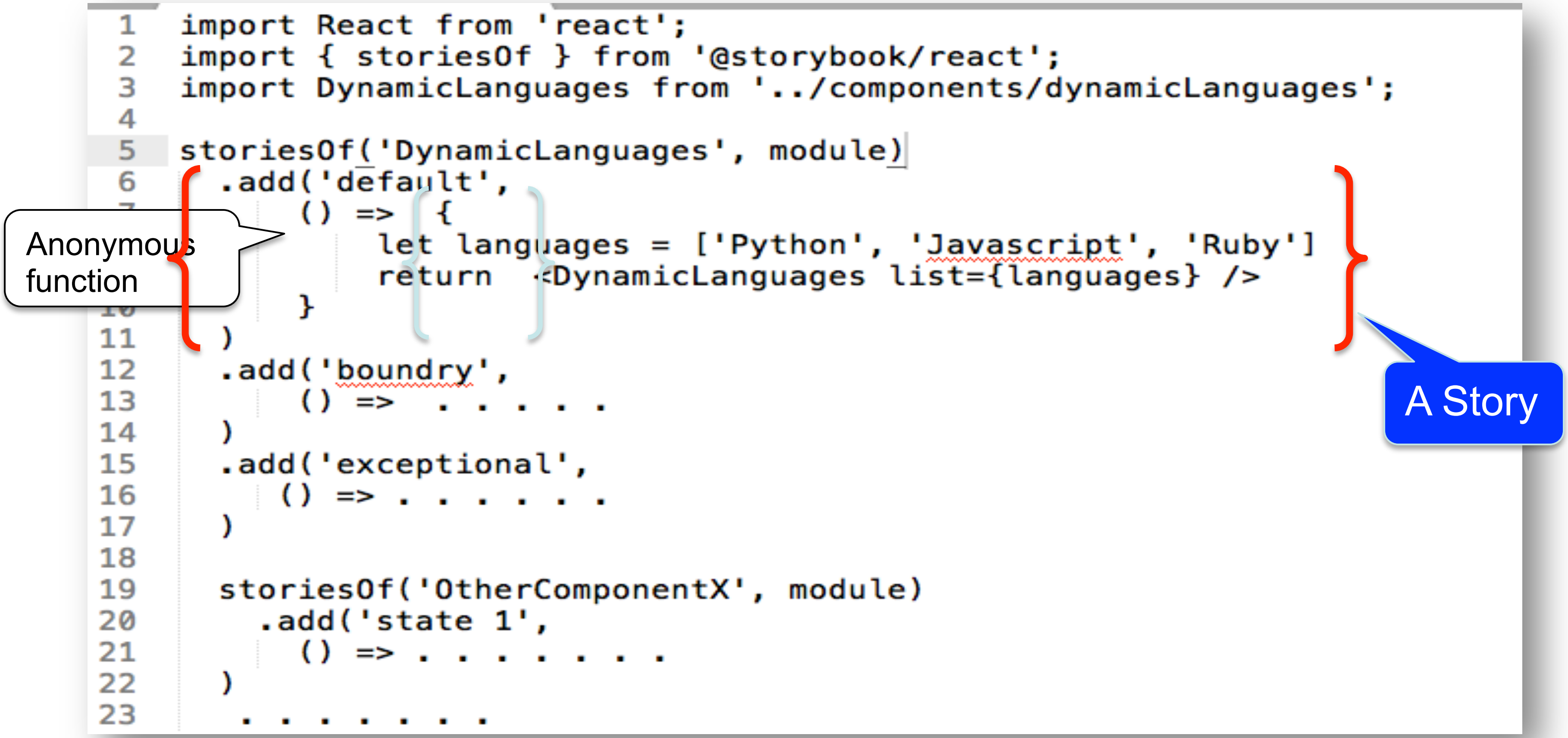
Dynamic Languages

- Python
- Javascript
- Ruby
- PHP
- Groovy

See more ...

Writing stories

- **Tool provides a (fairly) declarative syntax for writing stories.**
  - **Like a domain specific language (DSL).**

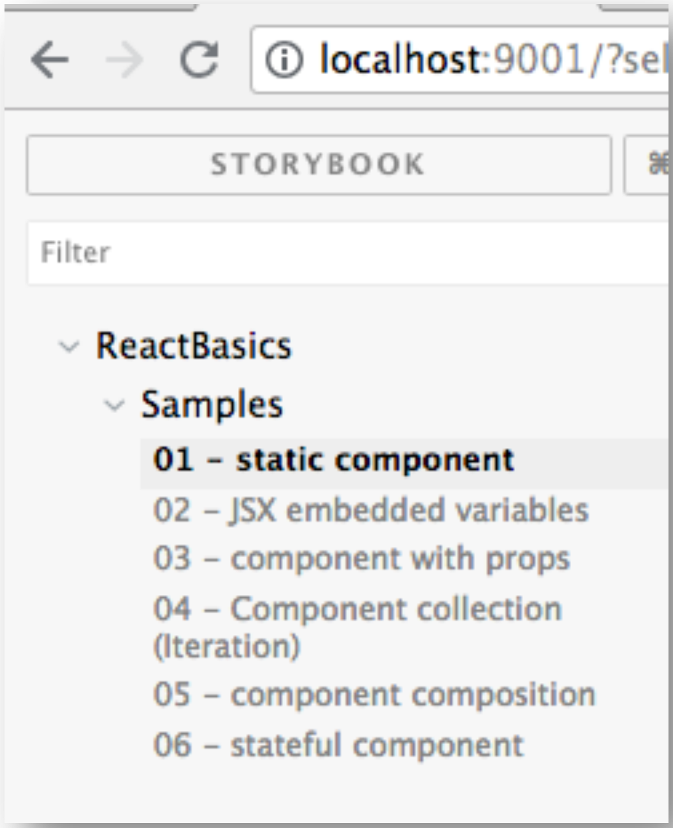


```
import React from 'react';
import { storiesOf } from '@storybook/react';
import DynamicLanguages from '../components/dynamicLanguages';

storiesOf('DynamicLanguages', module)
  .add('default',
      () => {
          let languages = ['Python', 'Javascript', 'Ruby']
          return <DynamicLanguages list={languages} />
      }
  )
  .add('boundry',
      () => . . . . . .
  )
  .add('exceptional',
     () => . . . . . . .
  )

  storiesOf('OtherComponentX', module)
    .add('state 1',
      () => . . . . . . .
  )
   . . . . . . .
```

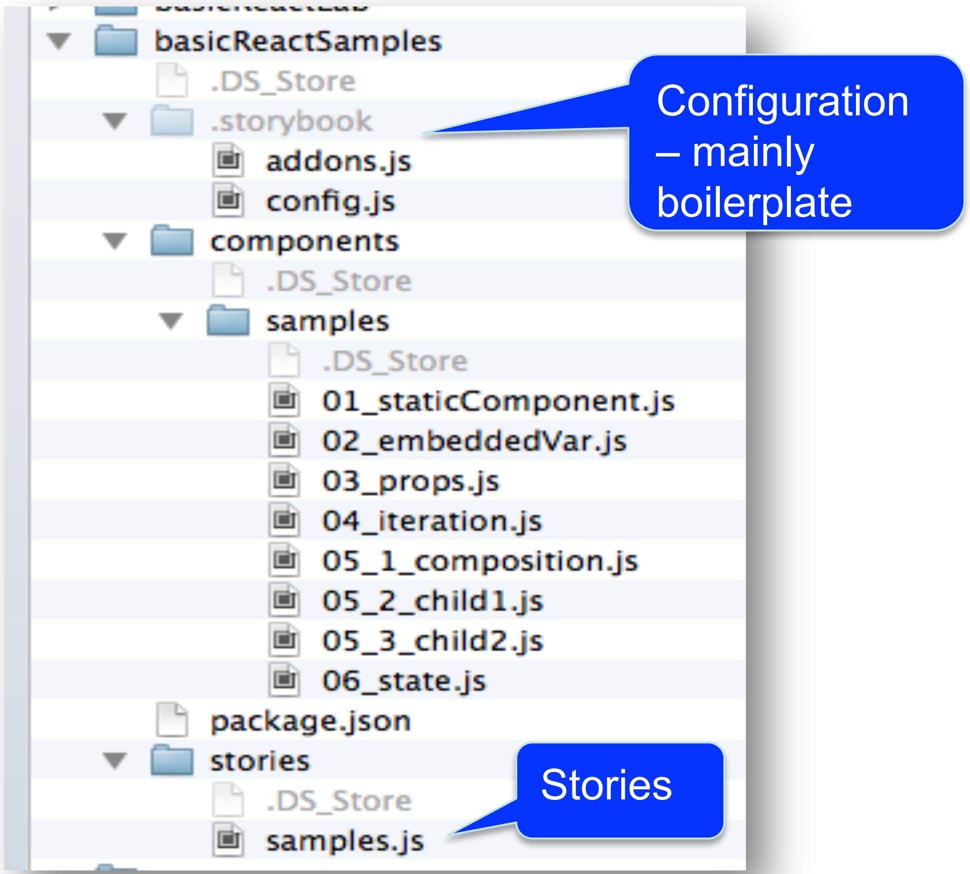
… back to components . . .

Samples

- **Samples to demonstrate Component features.**
  - **Basis for this week's lab..**



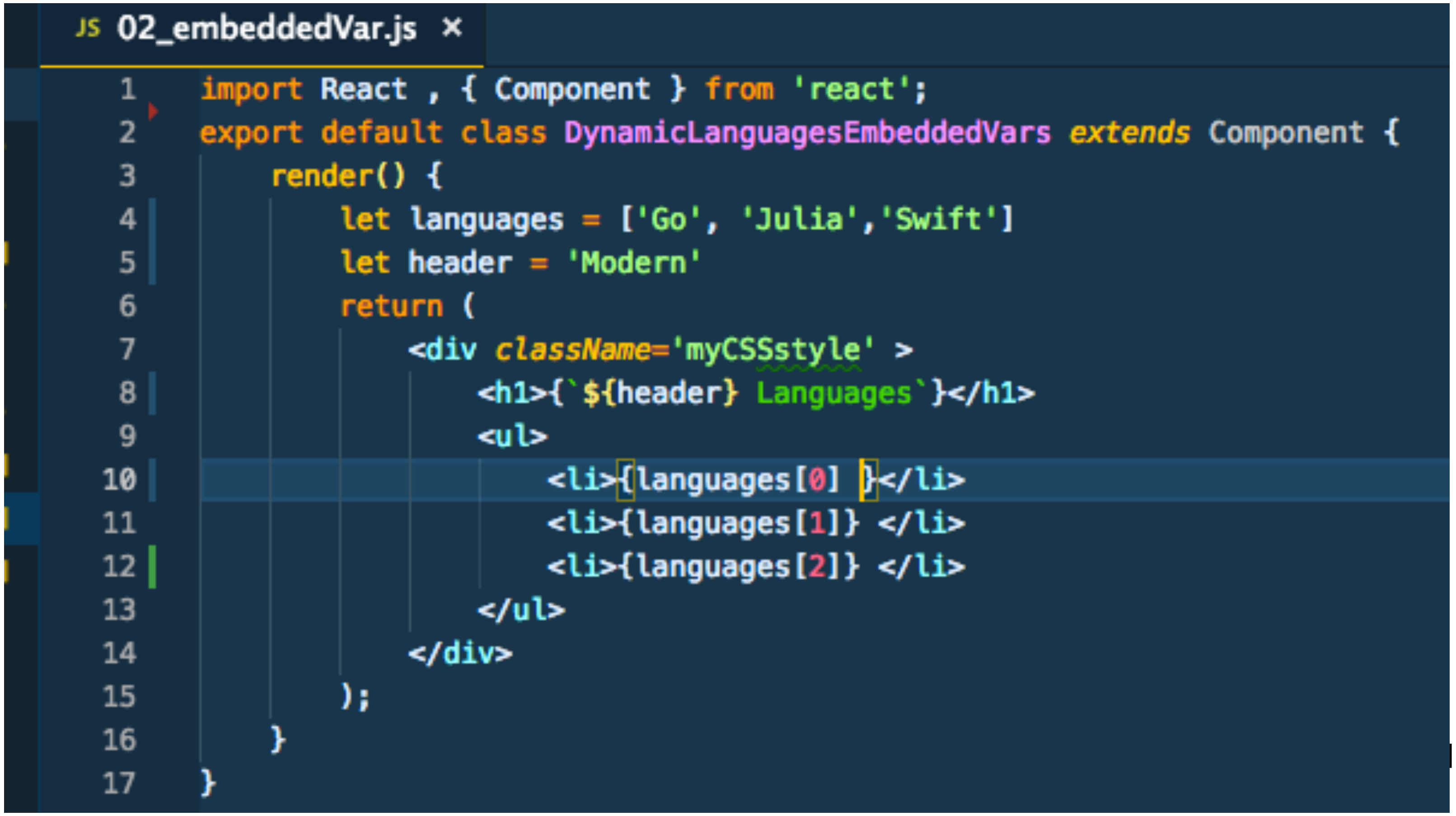
ReactBasics
- Samples
  - **01 – static component**
  - 02 – JSX embedded variables
  - 03 – component with props
  - 04 – Component collection (Iteration)
  - 05 – component composition
  - 06 – stateful component



```
basicReactSamples
  .DS_Store
  .storybook
    addons.js
    config.js
  components
    .DS_Store
    samples
      .DS_Store
      01_staticComponent.js
      02_embeddedVar.js
      03_props.js
      04_iteration.js
      05_1_composition.js
      05_2_child1.js
      05_3_child2.js
      06_state.js
  package.json
  stories
    .DS_Store
    samples.js
```

JSX embedded variables.

- **Dereference variable embedded in JSX code using { } braces.**
  - **Braces can contain any valid JS expression.**
- **Reference** 02_embeddedVariables.js



02_embeddedVar.js

```
import React , { Component } from 'react';
export default class DynamicLanguagesEmbeddedVars extends Component {
    render() {
        let languages = ['Go', 'Julia','Swift']
        let header = 'Modern'
        return (
            <div className='myCSSstyle' >
                <h1>{`${header} Languages`}</h1>
                <ul>
                    <li>{languages[0] }</li>
                    <li>{languages[1]} </li>
                    <li>{languages[2]} </li>
                </ul>
            </div>
        );
    }
}
```

Reusability.

- **Achieve reusability through** parameterization**.**
- props **– Component properties/attribute;**
  - **Passing props to a component:**

    <CompName prop1Name={value} prop2Name={value . . . . />
  - **Access inside component via** this.props **object:**

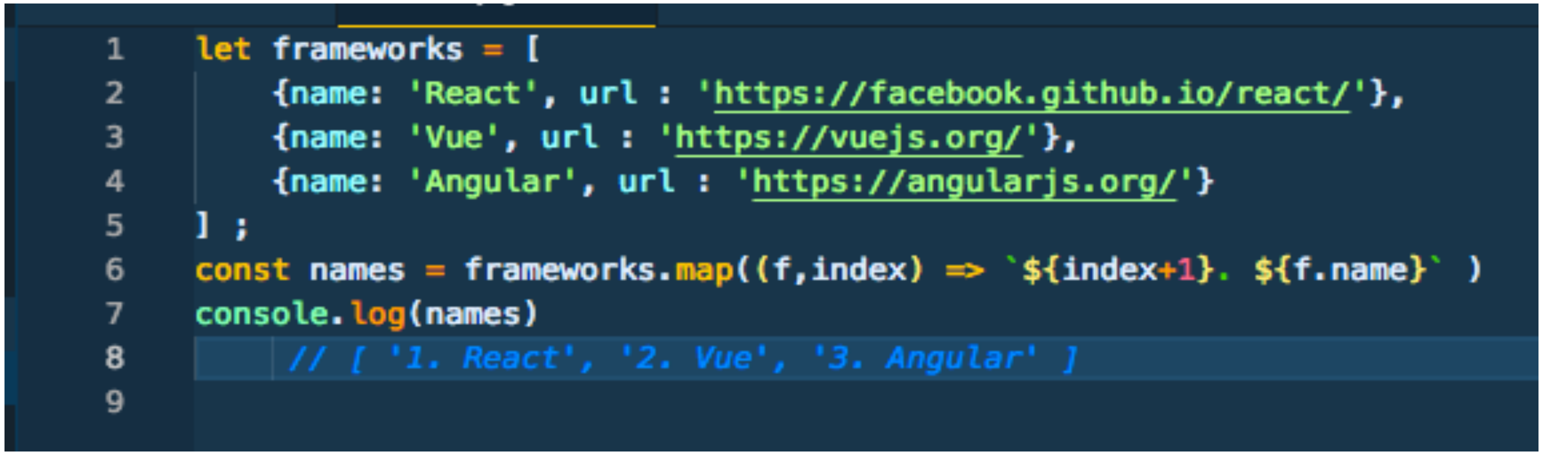
    let p1 = this.props.prop1Name
  - **Immutable.**

- **Reference** 03_props.js **and related story.**

Aside – Some JS issues

- **When an arrow function has only ONE statement, which is its return value, then you may omit:**
  - **Body curly braces; ‘return’ keyword; Semi-colon**

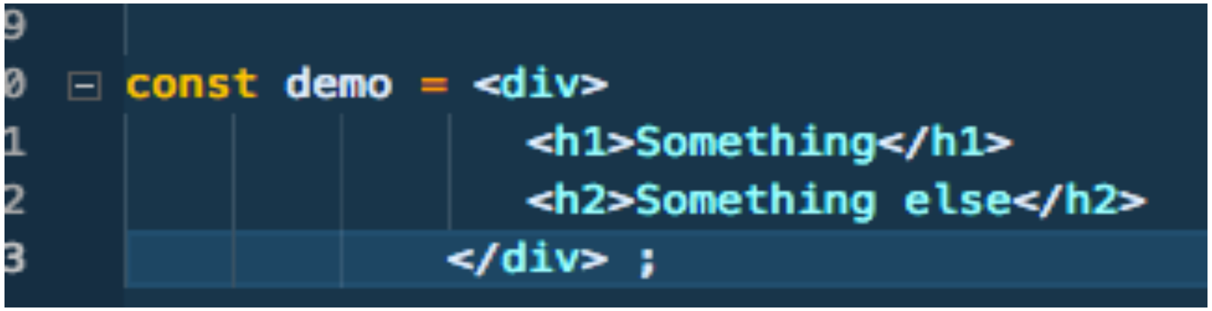
- **The Array** map **method – returns a new array based on applying the function argument to each element of the source array.**



```
let frameworks = [
    {name: 'React', url : 'https://facebook.github.io/react/'},
    {name: 'Vue', url : 'https://vuejs.org/'},
    {name: 'Angular', url : 'https://angularjs.org/'}
] ;
const names = frameworks.map((f,index) => `${index+1}. ${f.name}` )
console.log(names)
    // [ '1. React', '2. Vue', '3. Angular' ]

```

Aside – Some JS issues

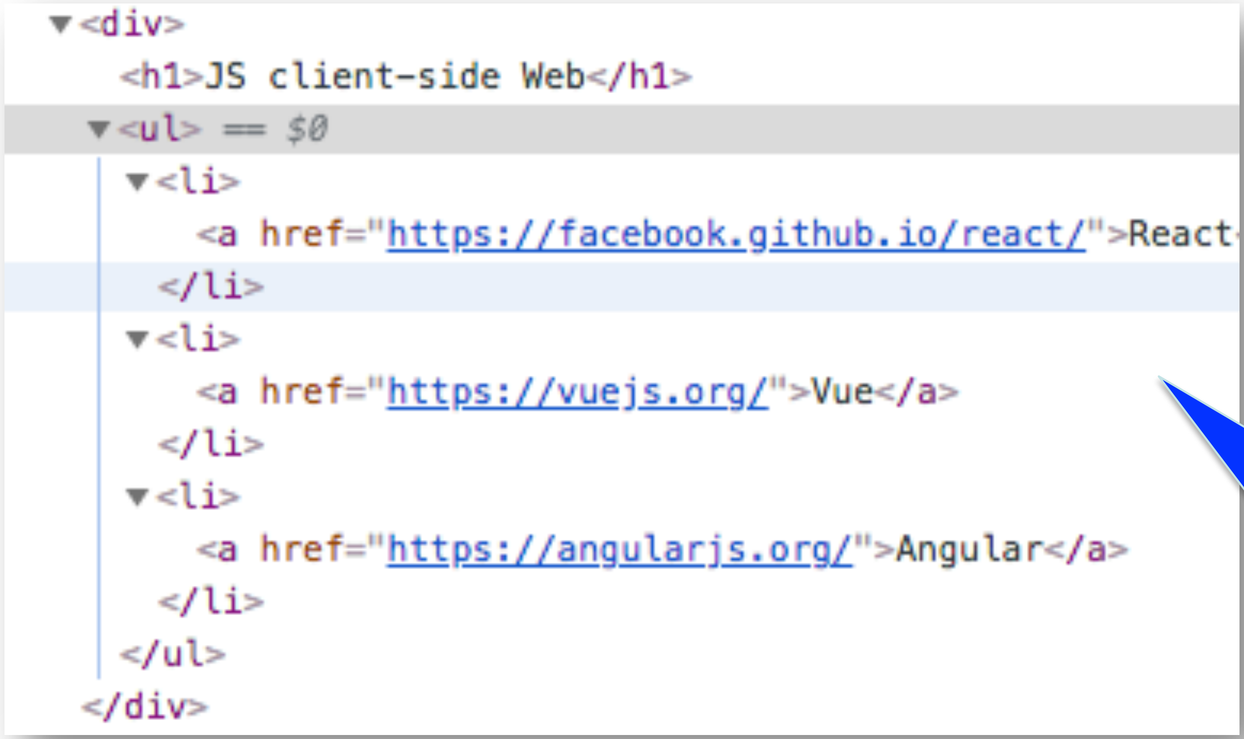
- **We can assign a** single **JSX element to a variable.**



```
const demo = <div>
                <h1>Something</h1>
                <h2>Something else</h2>
             </div> ;
```

Component collection - Iteration

- **Obj.: Generate a collection of component instances.**
- **Reference** 04_iteration.js



```
▼<div>
   <h1>JS client-side Web</h1>
 ▼<ul> == $0
   ▼<li>
      <a href="https://facebook.github.io/react/">React
    </li>
   ▼<li>
      <a href="https://vuejs.org/">Vue</a>
    </li>
   ▼<li>
      <a href="https://angularjs.org/">Angular</a>
    </li>
   </ul>
 </div>
```

Real DOM produced by story (From Chrome Dve Tools)

The render() return value.

- **Examples:**
  - return <h1>Something</h1> ;
  - return <MyComponent prop1={…..} prop2={……} /> ;
  - return (

```
        <div>
          <h1>{this.props.type}</h1>
          <ul>
              . . . . . . .
          </ul>
        </div>
  ) ;
```

- **Must enclose in ( ) when multiline.**

The render() return value.

- **Must return only ONE element.**
- **Examples:**

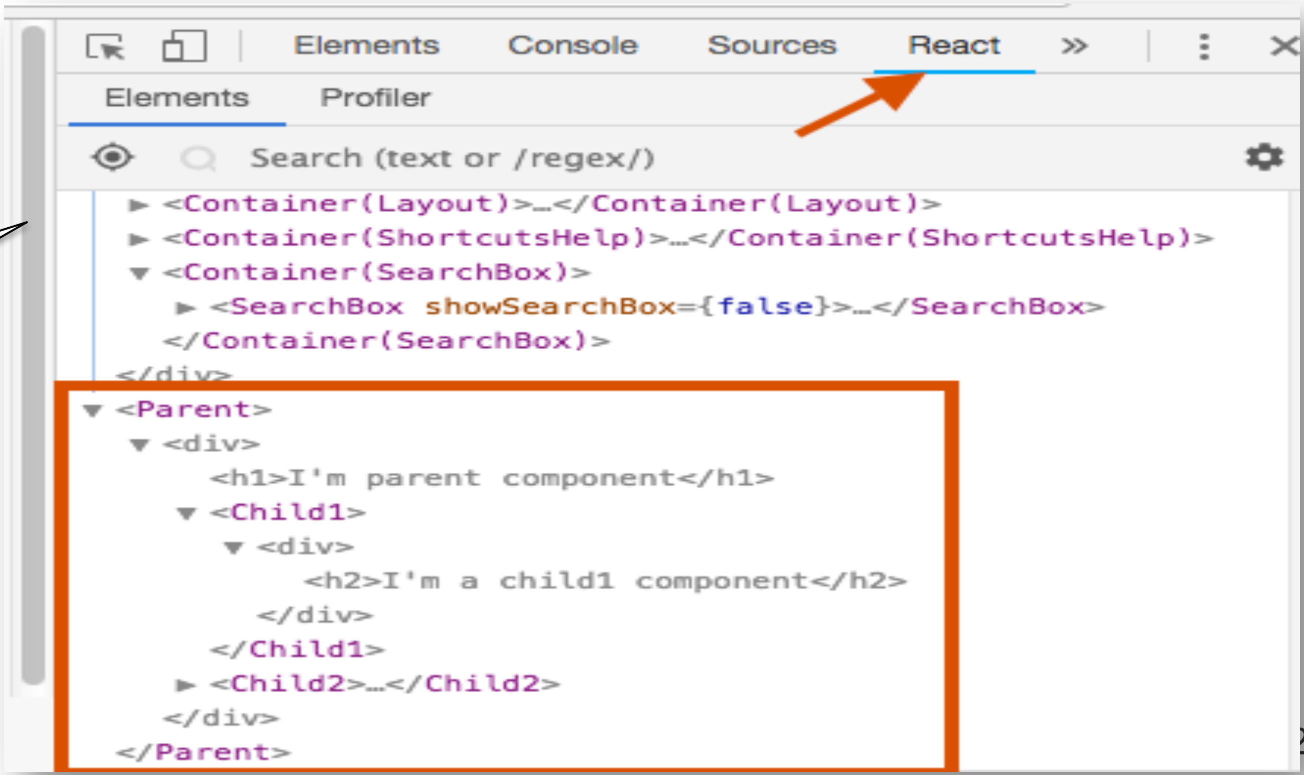
  -
    ```
    return (
            <h1>{this.props.type}</h1>
            <ul>
                  . . . . . . . .
            </ul>
       ) ;
    ```
  - **Error** – ‘Adjacent JSX elements must be wrapped in an enclosing tag**’**
  - **Solution: Wrap in a div tag.**

Component ***Composition***.

A React application is designed as a hierarchy of components.

- **Components have** children **– nesting.**
- ***See*** *05_1_composition.js.*

From Chrome Dev Tools React extension



```
▶ <Container(Layout)>…</Container(Layout)>
▶ <Container(ShortcutsHelp)>…</Container(ShortcutsHelp)>
▼ <Container(SearchBox)>
  ▶ <SearchBox showSearchBox={false}>…</SearchBox>
  </Container(SearchBox)>
</div>
▼ <Parent>
  ▼ <div>
      <h1>I'm parent component</h1>
    ▼ <Child1>
      ▼ <div>
          <h2>I'm a child1 component</h2>
        </div>
      </Child1>
    ▶ <Child2>…</Child2>
    </div>
</Parent>
```

Summary.

- **Exam**

Component DATA

- **Two sources of data for a component:**
  - Props **- Immutable; Passed in by parent component.**
  - State **– Dynamic; Managed internally by the component**

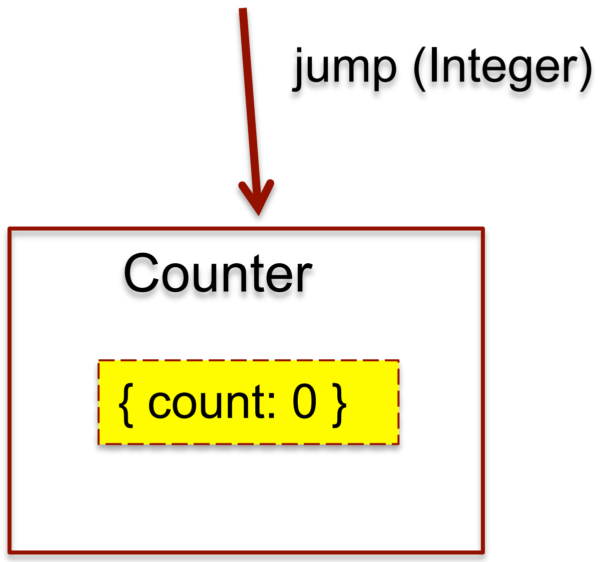
- **State-related component features:**
  1. I**nitialize state values.**
  2. **Change the state values- the** setState**() method.**
     - **Change some or all state values (merge, not overwrite).**
     - **Automatically causes component to re-render. *****

- **Props-related component features:**
  1. Set d**efault prop values.**
  2. **Type-checking**.

Making components interactive and dynamic

- **Ex:The** Counter **component.**
- **Ref.** 06_state.js

- **Component class coding features:**
  1. **Custom functions, e.g.** incrementCount().
  2. **Static class property, e.g.** defaultProps.
  3. **Class instance property, e,g.** state.



React’s event system.

- **Cross-browser support.**
- **Event handlers receive** SyntheticEvent **– a cross-browser wrapper for the browser's native event.**
  - **Same interface as native event**
- **Event naming convention slightly different from native:**

React	Native
onClick	onclick
onChange	onchange
onSubmit	onsubmit

- **See https://reactjs.org/docs/events.html for full details,**

Re-rendering

- **EX.: The Counter component.**

  User clicks 'increment' button

    → onClick event handler (incrementCounter) executed

      → state is changed (setState())

    → render() method executed

The Virtual DOM

- **Traditional** performance **best practice:**
  1. **Avoid expensive DOM operations.**
  2. **Minimize access to the DOM.**
  3. **Update elements offline before reinserting into the DOM.**
  4. **Avoid tweking layouts in Javascript.**

- **Should the developer be responsible for low-level DOM optimization? Probably not.**
  - **React solution is the** Virtual DOM**.**
  - **A challenge to established thinking!**

The Virtual DOM

- **Re-render everything on every update.**
  - **Sounds expensive!**

- **How?**
  1. **Create lightweight description of app's UI (The Virtual DOM)**
  2. **Perform *diff* operation between it and the previous (virtual) UI state.**
  3. **Compute the minimal set of changes to apply to (real) DOM.**
  4. **Batch execute all updates to real DOM.**

- **Benefits:**
  a) **Clean – Clean, descriptive programming model**
  b) **Fast - Optimized DOM updates and reflows.**

Re-rendering (detail)

- **EX.: The Counter component.**

User clicks 'increment' button

→ onClick event handler (incrementCounter) executed

→ state is changed (setState())

→ render() method executed

*→ The **Virtual DOM** has changed*

→ React diffs the changes (between the current and previous Virtual DOM)

*→ React batch updates the **Real DOM***